virtual worlds

MDPI

Article

XR MUSE: An Open-Source Unity Framework for Extended Reality-Based Networked Multi-User Studies

Stéven Picard *, Ningyuan Sun and Jean Botev

VR/AR Lab, Department of Computer Science, University of Luxembourg, L-4364 Esch-sur-Alzette, Luxembourg; ningyuan.sun@uni.lu (N.S.); jean.botev@uni.lu (J.B.)
* Correspondence: steven.picard@uni.lu

Abstract: In recent years, extended reality (XR) technologies have been increasingly used as a research tool in behavioral studies. They allow experimenters to conduct user studies in simulated environments that are both controllable and reproducible across participants. However, creating XR experiences for such studies remains challenging, particularly in networked, multi-user setups that investigate collaborative or competitive scenarios. Numerous aspects need to be implemented and coherently integrated, e.g., in terms of user interaction, environment configuration, and data synchronization. To reduce this complexity and facilitate development, we present the open-source Unity framework XR MUSE for devising user studies in shared virtual environments. The framework provides various ready-to-use components and sample scenes that researchers can easily customize and adapt to their specific needs.

Keywords: extended reality; distributed systems; collaboration; interaction; software frameworks

Citation: Picard, S.; Sun, N.; Botev, J. XR MUSE: An Open-Source Unity Framework for Extended Reality-Based Networked Multi-User Studies. *Virtual Worlds* **2024**, *3*, 404–417. https://doi.org/10.3390/virtualworlds3040022

Academic Editors: Thiago Malheiros Porcino and Jorge C. S. Cardoso

Received: 9 August 2024
Revised: 16 September 2024
Accepted: 27 September 2024
Published: 2 October 2024

1. Introduction

With the growing proliferation and capabilities of immersive media technologies, extended reality (XR) systems have been increasingly employed in user and user-agent collaboration studies [1–4]. On the one hand, such systems allow for utilizing the full simulation potential of virtual reality (VR) to decouple users from reality and previous experience. On the other hand, the advantages of including computer-generated objects and behaviors can also be combined with familiar mechanisms from the real environment, as in Mixed Reality (MR) settings. The latter is particularly important for continuing to use a wide array of established communication methods between users in their usual form instead of laboriously recreating them. The use of XR in research generally offers many advantages, for example, in terms of controllability and consistency [5]. In addition to allowing entirely new forms and explorations of multi-user collaboration, scenarios that would otherwise pose a serious risk, such as emergencies [6] and disasters [7], can also be easily replicated.

However, creating such XR environments can be challenging as it requires numerous technical skills, such as 3D development, user interface (UI) design, network programming, and other XR-specific knowledge, to ensure high presence and immersion levels. Even for experienced developers, it would be time-consuming and labor-intensive, as the various aspects must be individually implemented and coherently integrated. To streamline the development process for researchers, we introduce XR MUSE (https://github.com/vrar-lab/XRMUSE, accessed on 16 September 2024), an open-source Unity framework for XR-based multi-user collaboration studies. The framework comprises several dedicated modules for different functionalities, including data synchronization, spatial calibration, or environment configuration, and also provides an example scene researchers can use and adapt to other contexts.

Before introducing the XR MUSE framework and specific concepts and features in Section 3, we will provide further background on XR-based collaboration and related

user studies, as well as existing toolkits for XR development, in Section 2. We conclude this article with a summary of the framework's key features and an outlook on the next development steps in Section 4.

2. Background

Over the last few decades, there has been a growing body of research on XR-based collaboration. The literature shows several unique benefits of XR-based collaboration compared to non-XR approaches. For users located in different places, collaborating in XR allows them to share a mediated reality in real time as if they were in the same space. This immersivity facilitates collaboration by providing a heightened sense of spatial presence and co-presence [8–10] as opposed to, for example, teleconferencing, where members see each other through screens. Furthermore, with XR, computer-generated objects can be included to create new ways of collaboration that otherwise would be difficult or infeasible in reality. To give an example, the design review meeting of an engineered system can be conducted in XR, so that the digital model of the system can be directly viewed, modified, and commented on in a shared space [11].

XR-based collaboration also benefits researchers as an experiment platform to investigate the dynamics underlying user–user or user–agent collaboration. Compared to physical reality, XR environments are more controllable since part of the environments are computer-generated. This allows researchers to repeat an experiment setup consistently across participants [5]. This consistency is advantageous for different types of studies, such as those involving confederates to create a specific collaborative scenario. Human actors' behaviors may fluctuate across sessions and, consequently, become an extraneous factor confounding the results [12,13]. Whereas in XR, collaborators can be replaced with virtual agents that can precisely replicate their behaviors following pre-defined scripts [14].

2.1. Technical Challenges in Collaborative XR Environments

Despite the above-mentioned advantages, creating a collaborative XR environment remains technically challenging since a complex of modules need to be individually implemented and coherently integrated. The following three modules, user interactions, spatial calibration, and networking, are common to most XR-based collaborative activities.

User Interactions. To maximize immersion and presence, XR-based applications usually allow users to grab, throw, and poke virtual objects by simulating or tracking users' hands. Compared to traditional WIMP-based (i.e., windows, icons, menus, and pointer) user interfaces, hand-based interactions feel more natural and intuitive [15] but involve more technical endeavors as they combines animations, physics simulation, computer vision, etc. The complexity further grows in multi-user scenarios, where a virtual object may be passed from one user's hand to another, during which the concept of ownership needs to be applied and synchronized. Furthermore, designing appropriate hand-based interactions can be demanding and knowledge-extensive. Numerous factors need to be optimized to maximize their positive effects on collaboration, ranging from static aspects such as the visual appearance of hands (e.g., shapes, skin tones, textures) to more dynamic ones like hand-tracking fidelities.

Spatial Calibration. When multiple MR devices are involved in an XR-based collaboration, they usually share the same physical space. However, each MR device has its own coordinate system to map the surrounding environment, and, consequently, the same coordinate may refer to different points in reality for different devices. To solve this issue, each individual mapping needs a unique transformation matrix to calibrate itself under a common coordinate system. While this calibration process can be boiled down to a mathematical problem, its implementation is not as straightforward. There are different approaches to spatial calibration, of which the most common one is by using fiducial markers such as QR or ArUco codes. The marker serves as a reference point for different MR devices to calculate the transformation between their spatial mappings. While being convenient, marker-based calibration is not persistent and requires re-calibration

if the application restarts. Cloud-based anchors (e.g., Microsoft Azure Spatial Anchors (https://azure.microsoft.com/services/spatial-anchors, accessed on 16 September 2024)) provide more persistent calibration by extracting feature points from the surrounding environment and storing them in the cloud. When the application is running, it automatically detects and compares feature points and calibrates its coordinate system accordingly.

Networking Networking is a fundamental and the most challenging part of creating collaborative XR environments. Most interactions in XR happen in real time and need to be constantly synchronized among different clients to sustain immersion. There are numerous libraries (e.g., Photon (https://www.photonengine.com, accessed on 16 September 2024), Mirror (https://www.github.com/mirrornetworking/mirror, accessed on 16 September 2024), Netcode (https://docs-multiplayer.unity3d.com, accessed on 16 September 2024)) that provide commonly used networking functionalities such as object synchronization (e.g., ownership, spawn, pooling), networked physics, and remote procedure calls (RPCs, an inter-process communication protocol to invoke subroutines in a remote client). Based on these libraries, developers can focus on building the application without having to handle low-level protocols and networking frameworks. However, despite the streamlined process, these libraries can still be challenging to use depending on developers' expertise and experiences. Specifically in scenarios involving XR-based interactions, they often need to be combined with other XR-related packages (cf. Section 2.2), which are natively built for single-user scenarios.

2.2. XR Development Toolkits

Various approaches exist to create an XR experience, through game engines (e.g., Unity, Unreal Engine, Godot Engine (https://godotengine.org, accessed on 16 September 2024), modeling software (e.g., Blender, Maya), or platforms (e.g., Nvidia Omniverse (https://www.nvidia.com/omniverse, accessed on 16 September 2024). Unity is a popular choice as a game engine due to its ease of use, numerous third-party plug-ins, and large user base. Developers can find a variety of useful toolkits for XR development in Unity, often built and supported by leading IT companies, such as the XR Interaction Toolkit (https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0, accessed on 16 September 2024) (Unity), AR Foundation (https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.0 accessed on 16 September 2024) (Unity), MRTK (https://github.com/mixedrealitytoolkit/mixedrealitytoolkit-unity, accessed on 16 September 2024) (Microsoft), Meta XR SDKs (https://developers.meta.com/horizon/documentation/unity/unity-package-manager, accessed on 16 September 2024), and others.

The widely used XR Interaction Toolkit provides a variety of versatile cross-platform components for managing XR inputs and interactions. These components cover a broad spectrum of functionalities such as locomotion, object manipulation (e.g., grab, throw), haptic feedback, and more. Developers can directly use this high-level interface without tailoring their code base to the APIs of individual target platforms. Similarly, the AR Foundation package is designed specifically for augmented reality development, wrapping commonly used functionalities (e.g., plane detection, face recognition) from vendors' APIs (e.g., ARKit (https://developer.apple.com/augmented-reality/arkit, accessed on 16 September 2024), and ARCore (https://developers.google.com/ar, accessed on 16 September 2024) into an overarching higher-level interface.

Some toolkits are provided by hardware vendors mainly for their own products. For example, Meta creates and maintains the Meta XR SDKs, primarily targeting its headsets. It contains basic XR support, such as locomotion and input management, and features specific to Meta's vision—e.g., lip sync, body tracking, and avatar assets—to create a social environment. There are also community toolkits developed by individuals or small commercials like VRIF (https://assetstore.unity.com/packages/templates/systems/vr-interaction-framework-161066, accessed on 16 September 2024) or HurricaneVR (https://assetstore.unity.com/packages/tools/physics/hurricane-vr-physics-interaction-toolkit-177300, accessed on 16 September 2024), which are usually built upon the ones mentioned above to

provide out-of-the-box but opinionated implementations for XR game development. The VRIF package, for example, allows users to climb ladders, grab and fire a gun, and more.

Overall, these different toolkits provide common functionalities such as locomotion, object interaction, and more, significantly facilitating XR development. However, creating a shared XR experience based on these toolkits remains challenging. Most only provide fundamental features but do not wrap them into useful out-of-the-box components or provide a ready-to-use framework for behavioral studies. For instance, these toolkits do not include user connectivity and require developers to integrate third-party networking plugins into their applications.

2.3. Existing Frameworks for XR-Based Collaboration

The literature has documented several frameworks (e.g., [16–20]) for creating a shared collaborative space, to which users can connect via the device of their preference, including but not limited to VR headsets, MR glasses, hand-held AR devices, tablets, and desktops. These frameworks share similar technology stacks—with Unity and a subset of the above-mentioned toolkits—but target different scenarios with their individual implementations.

For example, most of the frameworks employ a server–client structure to connect different XR devices. Some of them are built upon existing solutions such as Unity Netcode or Photon [18–20], while others implemented their own server–client communication using, for instance, the UDP protocol [17]. The latter approach is more challenging but provides full control of the server logic. This liberty in server design can be advantageous for experiments on XR-based collaboration, since researchers can monitor all the clients and collect their data simply on the server side. Other than the server–client structure, one study opted for a peer-to-peer structure to allow for collaboration without a reliable network connection [16].

The aforementioned frameworks were all demonstrated to effectively synchronize different XR devices. However, none of the frameworks is open-source, leaving other researchers mostly theoretical insights without a usable code base. XR MUSE stands out from these existing frameworks for being publicly accessible on GitHub, providing not only several ready-to-use packages but also an example scene to work with. There also exist other open-sourced frameworks, such as the Unity Experiment Framework [21] and the BiomotionLab Toolkit for Unity Experiments [22]. However, they mainly target behavioral studies in general rather than focusing on XR-based collaboration.

Outside academia, various commercial products allow users to interact and collaborate in a shared XR space. Many are video games (e.g., Cook-Out (https://www.resolutiongames.com/cookout, accessed on 16 September 2024)), where players must act toward common goals in dedicated, pre-defined environments. Content-centric platforms, where users create objects and experiences, such as VRChat (https://hello.vrchat.com/, accessed on 16 September 2024) and Rec Room (https://recroom.com/, accessed on 16 September 2024), offer a higher level of customization, allowing users to create multi-user XR-based environments via APIs. While such commercial products and platforms can also be used for observational or field studies on multi-user collaboration in XR [23], running controlled experiments with them is challenging. Commercial products are usually optimized for socializing or entertainment and contain various elements (e.g., eye-catching animations) that are not controllable or are irrelevant to the experiment goal. These elements may distract participants or confound experiments, potentially leading to validity issues. In comparison, as an open-source solution, XR MUSE offers a minimal, neutral environment that researchers can fully control.

3. XR MUSE

The XR MUSE framework consists of the following four central packages that are based on existing and widely used technologies, as illustrated in Figure 1:

- *XRMUSE.Networking* (cf. Section 3.2) contains a collection of scripts for networking-related functionalities, including server connection, data synchronization, object pool-

ing, networked physics, and more. At this point, these scripts are built upon the Photon Unity Network (PUN2 (https://assetstore.unity.com/packages/tools/network/pun-2-free-119922, accessed on 16 September 2024)), which uses the free remote server to connect XR devices.

- *XRMUSE.SceneDescription* (cf. Section 3.3) provides a system that allows researchers to describe the environment with text files following a specific syntax. When the application starts, these files are loaded and parsed by the application to instantiate the environment. To change the environment's parameters, only the configuration files need to be modified instead of the full scene within Unity. This dynamic loading system not only provides a convenient and flexible way to contextualize and adapt the environment but also hides the technical details behind scene creation in Unity.
- *XRMUSE.Utilities* encapsulates other generic functionalities that do not fit the packages mentioned above, like custom collisions (cf. Section 3.1.1) or spatial calibration.
- *XRMUSE.ExampleScene* (cf. Section 3.1.2) provides an example scene where two users work collaboratively to accomplish an industrial assembly task. The package includes various assets used for building the scene, such as meshes, textures, and materials, and a batch of scripts for realizing the logic behind the scene. The key elements in the scene are modularized into individual Unity Prefabs, which are reusable assets that can store a so-called GameObject along with its components and properties. The scene is deliberately kept generic to offer a baseline environment that can be easily customized for other collaborative tasks.

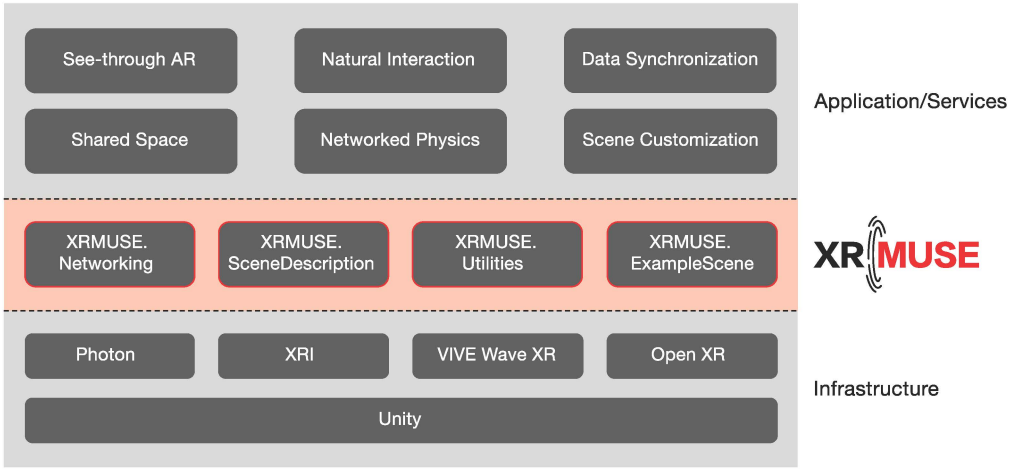

Figure 1. XR MUSE framework structure. The packages act as an intermediate between the infrastructure (i.e., common toolkits for XR development) and application (XR-based collaboration) layers.

3.1. Environment

XR MUSE features a workbench setting as an example scenario, where users share and interact with virtual objects on a real table. Figure 2 illustrates this setting in a dual-user case, where the physical space is conceptually divided into shared and individual zones. The area on and around the workbench is designed to be the main collaboration space. In contrast, the areas to the left and right are accessible only to the respective user.

This central workbench setting enables organic interaction between users within the designated scenarios. It is a particularly practical arrangement since any table can be used, and the physical separation between users avoids risks associated with co-location. As illustrated in Figure 3, users can access different resources either shared between them to collaborate, or only accessible to an individual for competitive advantage or informational asymmetry. This could be both tangible (e.g., the cubic objects on the table surface or the different tools of the users) or intangible (e.g., the sink seemingly altering the table's geometry or the information panel, which here is visible only to one of the users).

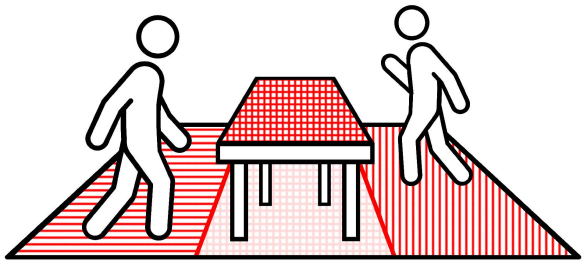

Figure 2. Conceptual division of the example environment. The table surface (checkered) is shared, whereas the left (horizontally ruled) and right (vertically ruled) areas belong to individual users.

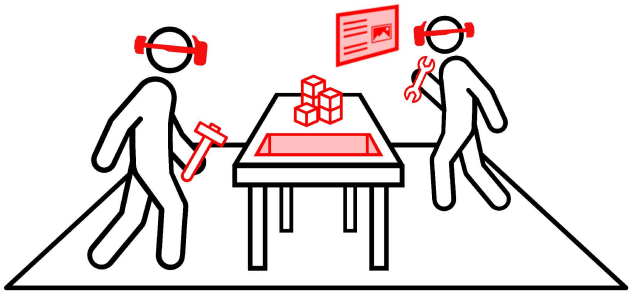

Figure 3. Digital content in XR includes objects and tools, as well as intangible items like virtual cavities or information panels that might be shared between or accessible only to individual users.

This setup assumes the users' individual space is in the same coordinate system that aligns with physical reality. For calibration, XR MUSE offers a headset-specific implementation. Once calibration is completed, the virtual objects will rest accurately on the table. Users can grab, release, and throw them using controllers. To make the user experience seamless, all three actions are bound exclusively to the so-called grip or trigger buttons, leaving the other buttons unused. This facilitates operation and significantly flattens the learning curve for employing the controllers, allowing users to quickly master the interaction regardless of their technical prowess and prior knowledge. This is essential for experiments and user studies, as changing experiences with the controllers can distort the results, and participants might be bored or frustrated by a lengthy familiarization process. All interactions in the example environment are collision-based and utilize virtual objects.

3.1.1. Collision System

To allow in-environment objects to interact with each other, XR MUSE provides a typed collider system on top of Unity's collision system. Each typed collider has an identifier of a custom type, which hints at the nature of the collision (in the example scenario, e.g., a material with another material or with a tool). Developers can add custom behaviors to these colliders that react to the type of colliding or existing objects. For instance, in the example scenario, two colliders typed as material colliding with each other will trigger a glow animation; when these colliders exit each other, the animation is toggled off (cf. Figure 4). In conjunction with the dual-transform design (cf. Section 3.2.1), this system allows one to dissociate the Unity physics from custom collision-based interactions.

(**a**) (**b**)

Figure 4. Example usage of custom collisions triggering the material outline for objects in proximity. (**a**) Screenshot, custom colliders set to invisible. (**b**) Screenshot, custom colliders set to visible.

3.1.2. Example Scene and Setup

Similar to the illustration in Figure 3, two users are tasked to produce a certain amount of products using the provided materials in the example scene. Each user has a unique tool to combine two materials into an intermediate or final product. Several combinations are possible, with each user only capable of realizing a subset, so they have to work collaboratively to produce the final product. All the materials and products are represented abstractly as basic primitives like cubes or spheres, whereas the tools have more specific visuals than wrenches or hammers. The virtual cavity on the table allows users to submit their products by throwing them into the cavity.

To combine two materials, users need to put them together and hit one of the materials with their tools, as illustrated in Figure 5. Internally, this is achieved through the typed collision system, where the collision between materials and tools triggers several networked events. These events handle and synchronize the task logic and animation of the combination process across users.

(**a**) (**b**)

Figure 5. Example material combination sequence, from two source objects to the final product. (**a**) Screenshot, hammering together source objects. (**b**) Screenshot, resulting target sphere object.

Apart from the interactive elements discussed above, there are also static information panels (cf. Figure 6) in the environment showing users the overall objective (i.e., which and how many products to produce), possible combinations, and the current status (i.e., how many products have been produced). As discussed in Section 3.1, these panels can be shared or only visible to an individual user. Informational asymmetry, as in the latter case, can be crucial to specific task types. For example, one user may see "two cubes with a hammer produce a sphere", while the other sees "a cube and a sphere with a wrench produce a cylinder". The objective in this case can be set to produce five cylinders, such that the two users must communicate and work collaboratively to reach the goal.

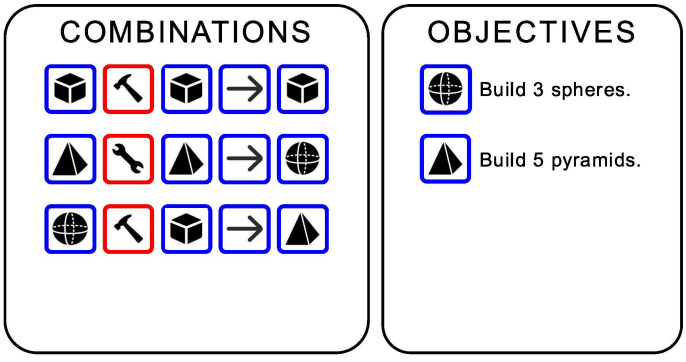

Figure 6. Outline view of the panels in the example scene with available combinations and objectives.

In the example setup, we employ two VIVE XR Elite (video-see-through mixed-reality headsets) to maximize the perception of presence and immersion. The choice of devices, however, is flexible and can also include virtual reality headsets such as the Meta Quest 3. The two headsets are connected through a remote third-party server via the internet. The communication among headsets follows a customizable protocol (cf. Section 3.2) built to facilitate experiment design. This introduces a small but acceptable latency, which can be further reduced by setting up a local server. This example was deployed using Unity

2022.3.16f1 with the support of numerous packages, including both commonly used ones such as XR Interaction Toolkit and dedicated packages for networking (e.g., Photon) and XR development (e.g., VIVE Wave XR Plugin). Along with the example Unity project, a batch of configuration files are also provided (cf. Section 3.3), allowing researchers to easily customize the example without the need to modify the Unity project directly.

The users' virtual spaces are calibrated to the real-life table using the XR Elite's ArUco marker detection feature. Experimenters can simply place a marker at the center of the table, which then serves as a reference point.

3.2. Networking

Networking is an essential but technically challenging aspect of multi-user XR environments. The XR MUSE framework reduces the complexity by providing a ready-to-use, configurable networking module developed based on the Photon Unity Network (PUN2) for ease of use and similarity to UNet. It should be noted that the network solution requires a platform-specific implementation involving concrete code and design, which will be briefly discussed before proceeding to XR MUSE's other systems. In developing the framework, we considered the PUN2 requirements that (1) a Unity GameObject is authored (or owned) by a user and ownership may be transferred between users, and (2) each networked GameObject may synchronize its data through de/serialization. Building on these two concepts, the module provides specific implementations for ownership transfer and transform synchronization, as well as a pooling system for GameObject instantiation and recycling. These specific implementations, which are required for using PUN2, are not the focus of this article and are therefore not discussed in detail. However, they are necessary foundations for the following subsections on the dual-transform design (Section 3.2.1) and data synchronization (Section 3.2.2).

XR MUSE was developed with co-located shared environments in mind, i.e., mixed-reality settings where users share a physical and digitally augmented space. The possibility of also interacting in a non-mediated fashion is an essential aspect of collaborative studies. However, the framework's networking components already support completely remote scenarios where users can be anywhere. In such cases, only an audiovisual representation of the users with an avatar, including voice, needs to be added.

3.2.1. Dual-Transform Design

In the virtual environment, users manipulate physical objects by moving them, placing them on the workbench or elsewhere, and interacting with other objects. In a networked environment, however, other users can also influence these objects. For example, physical interactions (e.g., gravity) can overlap with another user's actions. In Unity, physical objects are usually divided by a hierarchy of elements, and collisions can occur on an element at a higher point in the hierarchy. Each element of this hierarchy has a transformation that determines the positions of the subordinate elements.

In XR MUSE's dual-transform design (cf. Figure 7), two objects with a parent–child relationship in the same hierarchy are considered to be on the same level, with the authoring determined by one of the two transformations. The parent transform typically is the networked GameObject; its transform is synced, and when the user does not own the object, that parent transform is in charge of the dual-transform's authoring. The child transform is not synced but takes the dual-transform's authoring when the user owns the synced GameObject, typically having the physics, local controls, and gestures over the object (cf. Figure 7).

This allows the dissociation of the synchronizing behavior from the local interactions and assigning unity physics computation only to the owning user. Since both transforms can have different colliders, the parent transform also uses the custom collision system (cf. Section 3.1.1), which enables the synchronization of data in the case of collision-based interactions on networked objects.

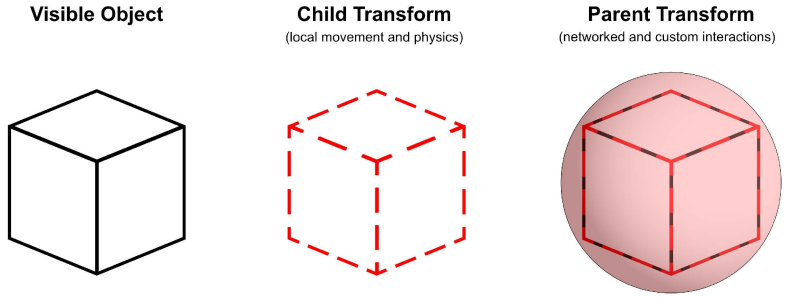

Figure 7. Example dual-transform design with the typed collision system on the parent.

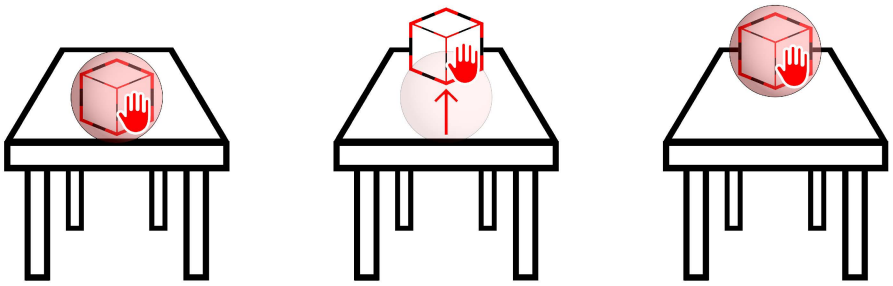

Figure 8. User moving object through child transform, parent transform snapping to new position.

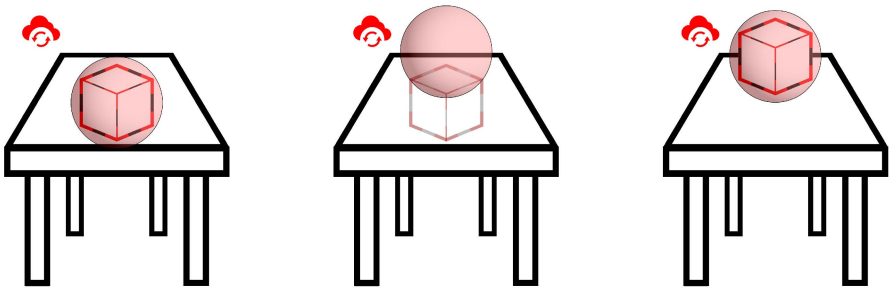

Figure 9. Network update of parent transform's position, child transform snapping to new position.

In the examples shown in Figures 8 and 9, the parent object uses XR MUSE's custom collision management system that does not trigger physical interactions. The geometrically simpler sphere of the parent transform is deliberately larger in diameter and completely encloses the actual object, enabling the following two aspects: first, separating the interactions between objects from the physics, which allows it to abstract tasks from the Unity engine's physics system, and second, activating physical authoring when a user needs the network object (e.g., when grabbing the object). Since only the position and rotation of the real parent are synchronized over the network, the networking does not affect the physics.

3.2.2. Data Synchronization

Not all synchronization needs fall into the category of GameObject synchronization. In an experiment, for instance, it may be necessary to synchronize environment states and record per-user actions. Therefore, XR MUSE includes a class created per user that shares its user-specific data with each connected user. The aggregation of these per-user data permits the deduction of environment states. For instance, if the objective is to produce together a certain amount of materials, this class will record the amount of materials each user produced, the aggregation representing the progress of this task (cf. Figure 10).

When using PUN or similar services, the network design assumes synchronization and serialization per virtual object, and different RPCs can be utilized to trigger events. This can be problematic when new data are added, where a new network object is created on both computers (or an existing object is edited). The calls must be triggered according

to the chosen network solution, which requires the associated ownership and authoring permissions. This system leverages this need for RPCs as local scripts can trigger events in response to updated values on the remote users' dedicated objects. Figure 10 shows an animation in reaction to one user's synchronized value of the CubeAnimation variable.

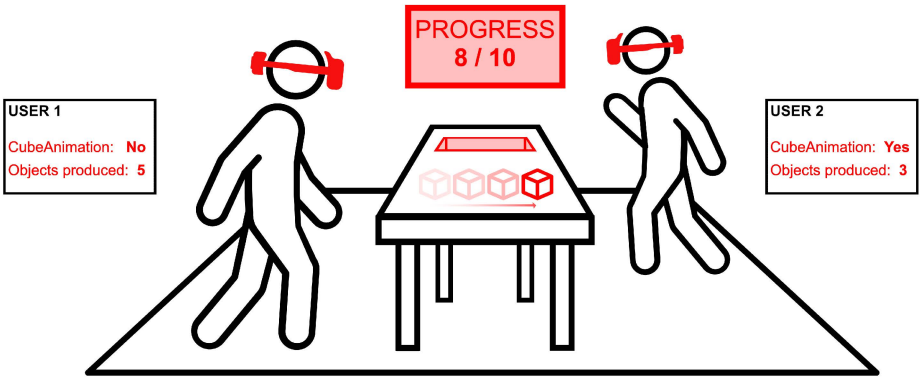

Figure 10. Aggregated per-user values affecting the scene for both users (progress display, animation).

More concretely, these user-specific data are created through a dedicated GameObject, to which we provide interfaces to append any custom type of data, which may also define accessible custom functions. That dedicated GameObject has a main component, *UserSyncedValues*, which handles the order of registered data, triggering serialization, and providing access to the various data and associated functions. This is realized using the C# programming language's reflection feature, which means that when loading, the object builds internal dictionaries of references to the corresponding exposed resources. Then, any number of user-defined *DataRegister* components that add data and functions for synchronization with the UserSyncedValues component can be added to it. The DataRegister components can mark each function as accessible via UserSyncedValues with a specific identifier (cf. diagram in Figure 11).

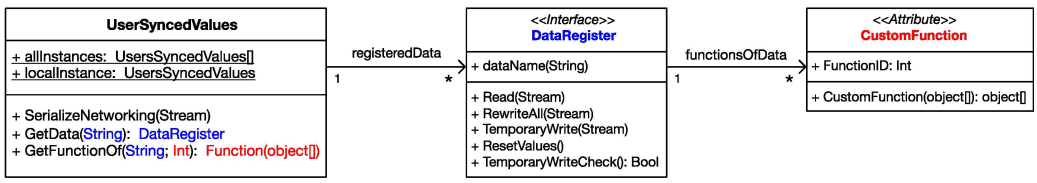

Figure 11. Simplified UML diagram for UserSyncedValues and DataRegister components reflection.

3.3. Operation

User studies [1–4] often create similar situations where only some elements of the environment or events are changed, e.g., to test a stimulus or collect data across several similar situations. While this can be achieved by creating multiple Unity scenes and special scripts for an experiment, the more scenes are added, the more complex maintenance becomes. Therefore, we propose a versatile scene loading system that has evolved from earlier work [24] towards a more generalized setting. This system is designed to adapt to a wide range of user studies, offering a solution that can be tailored to specific needs. The core elements of a scene can be defined in text files loaded at runtime from three types: instantiables, events, and stimuli.

3.3.1. Object Loader

An instantiable generates a template or Unity Prefab that other instantiables or events can use. Instantiables have an obligatory main type, which will set the main object generated but may also obtain optional sub-modules. An example of a set of instantiable files can be found in Figure 12, with declarations of the materials used in the scene and

boxes using these materials. These instantiables may be declared networked and compatible for automatic integration through XR MUSE's custom pooling system. A stimulus constitutes a similar template concept; however, it is explicitly separated from other instantiables as a stimulus may not be re-instantiated but activated or de-activated on demand. Implementation-wise, all declared instantiables are loaded by the module manager, which in the Unity scene can be observed with the creation of inactive Unity Prefabs, typically at scene load. Those inactive prefabs are the template than can be cloned by events described in the following subsection (Section 3.3.2).

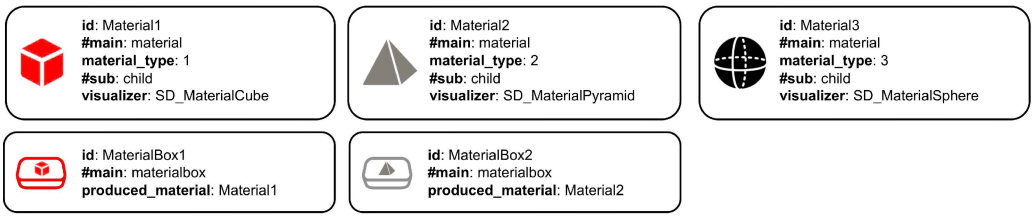

Figure 12. Example set of scene elements loaded from text files.

3.3.2. Scripting

Text files that are loaded at run-time also allow a form of scripting through a so-called events sequence. To handle the events sequence, XR MUSE uses a custom timeline, which also includes the production of log data. Event-specific logs may be produced during a trial except for the TIMELINE_LOAD and TIMELINE_START events. Events are either loaded from files or produced by other events They have an ID, a behavioral type (possibly custom), and a set of start conditions. The start conditions set is a collection of event-produced logs that all must be recognized. A log is a combination of an event's ID (or the default IDs for timeline load and start) and log type. The currently available log types are START, END, SUCCESS (which also produces END), and FAIL (which also produces END). Several generic events are available as examples and are sufficient for basic experiments, such as timers, instantiable spawning, or stimuli activation. Events can be declared as text files, just like instantiables and stimuli, with behavior-specific parameters that may refer to IDs of other declared files (such as an instantiable ID in the case of spawning instantiables) and the start condition logs. The example material boxes depicted in Figure 12 can be loaded by spawner events at loading, as shown in Figure 13.

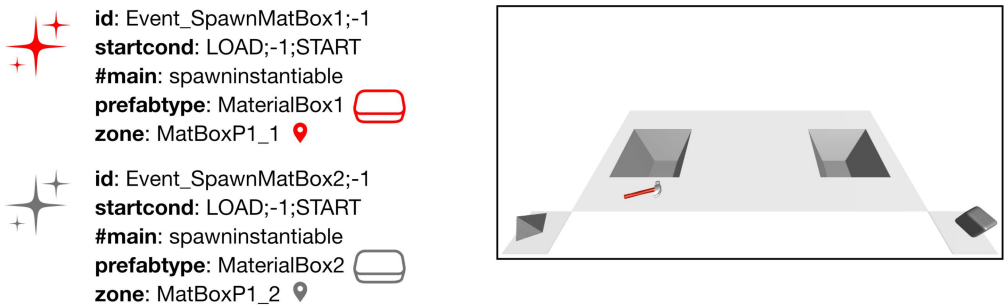

Figure 13. Scene after the TIMELINE_LOAD event.

As an example of more advanced scripting, a timer event may start with the automatic event TIMELINE_LOAD while another production task event has the success of the timer event as a starting condition (cf. Figure 14). When the timer generates its end log, the material production task can be started.

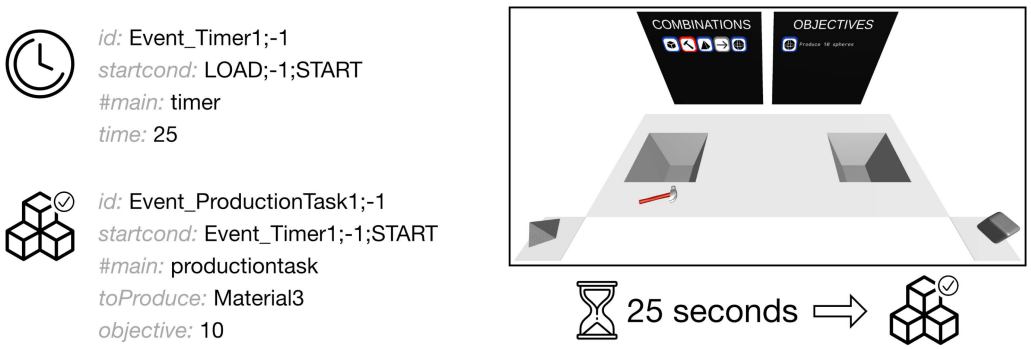

Figure 14. File-based definition of task start following a timer.

4. Summary and Outlook

This article introduces XR MUSE, an open-source framework that facilitates the creation of XR-based networked multi-user applications. Although it is designed for user studies in shared virtual environments, it can be used for any type of collaborative XR application. The framework significantly reduces some of the technical efforts by providing open solutions for dealing with XR-based interactions (Section 3.1.1), networking (Section 3.2), and scripting (Section 3.3), as well as a sample workbench environment integrating these elements (Section 3.1). Dedicated concepts, such as a custom collision system and dual transform design (Section 3.2.1), are provided to support real-time interaction. Implementing these together with the object loader (Section 3.3.1) unlocks the full potential of the corresponding XR MUSE packages.

Currently, the framework is based on Photon's PUN2, a free third-party Unity package that has some limitations due to its commercial nature. For example, XR MUSE uses Photon's free remote server for connecting different devices. Consequently, participants' personal data will go through their servers and risk exploitation. Although Photon offers a local server solution, it is not free. As a next step, we are thus considering porting to alternative open-source solutions such as Mirror.

At this stage, XR MUSE only supports two specific video see-through headsets (VIVE XR Elite and Meta Quest 3), which will be expanded in future versions of the framework to accommodate further or yet-to-be-released devices. Users also need to set up the devices by following relatively lengthy instructions; this setup process can be partially automated using scripts to improve XR MUSE's usability further.

It should be noted that XR MUSE was developed with functionality over performance in mind. Some of our advanced functionalities, like the scripting system (Section 3.3) and data registration (Section 3.2.2), which allow for extensive customization, may induce more processing than a purely custom solution. However, since most behavioral studies use minimal, neutral scenes for variable controls, the performance issues should be negligible. Still, in certain situations, such as when researchers try to load high-resolution assets, performance may suffer and affect the user experience. To improve XR MUSE, further stress tests with different devices are needed to identify potential performance issues.

XR MUSE's main application area is XR-based multi-user studies, in which users collaborate or compete in the same scene. For instance, we are currently employing the framework to conduct experiments on time perception in such scenarios. However, experimenters do not need to use the provided example scene. They may use only certain XR MUSE components, such as object synchronization, to handle distributed state. This is equally relevant for XR-based games or training environments. Application makers can use the framework as a prototyping tool, as the scene-loading features allow for the quick creation and testing of different scenarios with the same items. Different sets of configuration files can, for instance, represent various difficulties, including objects and their properties (e.g., physics and speed) to load as different stages and balance level the design quickly.

While XR MUSE is not a universal solution, it can be a valuable resource for diverse multi-user XR applications, either in its entirety or only in parts. The framework is freely available to anyone from researchers to XR application vendors. We look forward to potential collaborations and seeing XR MUSE as the basis for a new generation of customized multi-user XR software.

Author Contributions: Conceptualization, S.P. and J.B.; software, S.P., N.S. and J.B.; resources, J.B.; writing—original draft preparation, S.P., N.S. and J.B.; writing—review and editing, J.B.; visualization, J.B., S.P. and N.S.; supervision, J.B.; project administration, J.B.; funding acquisition, J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 964464 (ChronoPilot).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Mourtzis, D.; Angelopoulos, J. Development of an Extended Reality-Based Collaborative Platform for Engineering Education: Operator 5.0. *Electronics* **2023**, *12*, 3663. [CrossRef]
2. Lucas Bravo, P.P.; Fasciani, S. A Human-Agents Music Performance System in an Extended Reality Environment. In Proceedings of the 23rd International Conference on New Interfaces for Musical Expression (NIME), Mexico City, Mexico, 31 May–2 June 2023.
3. Pidel, C.; Ackermann, P. Collaboration in Virtual and Augmented Reality: A Systematic Overview. In Proceedings of the 7th International Conference on Augmented Reality, Virtual Reality and Computer Graphics, Lecce, Italy, 7 September 2020; pp. 141–156.
4. Santhosh, S.; De Crescenzio, F. A Mixed Reality Application for Collaborative and Interactive Design Review and Usability Studies. In Proceedings of the 11th International Joint Conference on Mechanics, Design Engineering & Advanced Manufacturing, Ischia, Italy, 1–3 June 2022; pp. 1505–1515.
5. Pan, X.; Hamilton, A.F.C. Why and How to Use Virtual Reality to Study Human Social Interaction: The Challenges of Exploring a New Research Landscape. *Br. J. Psychol.* **2018**, *109*, 395–417. [CrossRef] [PubMed]
6. Lorusso, P.; De Iuliis, M.; Marasco, S.; Domaneschi, M.; Cimellaro, G.P.; Villa, V. Fire Emergency Evacuation from a School Building Using an Evolutionary Virtual Reality Platform. *Buildings* **2022**, *12*, 223. [CrossRef]
7. Feng, Z.; González, V.A.; Amor, R.; Spearpoint, M.; Thomas, J.; Sacks, R.; Lovreglio, R.; Cabrera-Guerrero, G. An Immersive Virtual Reality Serious Game yo Enhance Earthquake Behavioral Responses and Post-Earthquake Evacuation Preparedness in Buildings. *Adv. Eng. Inform.* **2020**, *45*, 101118. [CrossRef]
8. Nguyen, H.; Bednarz, T. User Experience in Collaborative Extended Reality: Overview Study. In Proceedings of the 17th EuroVR International Conference, Valencia, Spain, 25–27 November 2020; pp. 41–70.
9. Reinhard, R.; Telatar, E.; Humayoun, S.R. Comparison of Object Detection in Head-Mounted and Desktop Displays for Congruent and Incongruent Environments. *Big Data Cogn. Comput.* **2022**, *6*, 28. [CrossRef]
10. Ochs, M.; Mestre, D.; De Montcheuil, G.; Pergandi, J.M.; Saubesty, J.; Lombardo, E.; Francon, D.; Blache, P. Training Doctors' Social Skills to Break Bad News: Evaluation of the Impact of Virtual Environment Displays on the Sense of Presence. *J. Multimodal User Interfaces* **2019**, *13*, 41–51. [CrossRef]
11. Sateei, S.; Roupé, M.; Johansson, M. Collaborative Design Review Sessions in Virtual Reality: Multi-Scale And Multi-User. In Proceedings of the 27th International Conference of the Association for Computer Aided Architectural Design Research in Asia (CAADRIA), Sydney, Australia, 9–15 April 2022; pp. 9–15.
12. Doyen, S.; Klein, O.; Pichon, C.L.; Cleeremans, A. Behavioral Priming: It's All in the Mind, But Whose Mind? *PLoS ONE* **2012**, *7*, e29081. [CrossRef] [PubMed]
13. Kuhlen, A.K.; Brennan, S.E. Language in Dialogue: When Confederates Might Be Hazardous to Your Data. *Psychon. Bull. Rev.* **2013**, *20*, 54–72. [CrossRef] [PubMed]
14. Koda, T.; Ruttkay, Z. Eloquence of Eyes and Mouth of Virtual Agents: Cultural Study of Facial Expression Perception. *AI Soc.* **2017**, *32*, 17–24. [CrossRef]
15. Kang, H.J.; Shin, J.H.; Ponto, K. A Comparative Analysis of 3D User Interaction: How to Move Virtual Objects in Mixed Reality. In Proceedings of the 27th IEEE Conference on Virtual Reality and 3D User Interfaces (IEEE VR), Atlanta, GA, USA, 22–26 March 2020; pp. 275–284.

16. Huh, S.; Muralidharan, S.; Ko, H.; Yoo, B. XR Collaboration Architecture Based on Decentralized Web. In Proceedings of the 24th International Conference on 3D Web Technology, Los Angeles, CA, USA, 26–28 July 2019; pp. 1–9.
17. Tümler, J.; Toprak, A.; Yan, B. Multi-user Multi-platform XR collaboration: System and Evaluation. In Proceedings of the 24th International Conference on Human-Computer Interaction, Virtual Event, 26 June–1 July 2022; pp. 74–93.
18. Kim, J.; Song, J.; Seo, W.; Ihm, I.; Yoon, S.H.; Park, S. XR Framework for Collaborating Remote Heterogeneous Devices. In Proceedings of the 1st IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Atlanta, GA, USA, 22–26 March 2020; pp. 586–587.
19. Pereira, V.; Matos, T.; Rodrigues, R.; Nóbrega, R.; Jacob, J. Extended Reality Framework for Remote Collaborative Interactions in Virtual Environments. In Proceedings of the 2nd International Conference on Graphics and Interaction (ICGI), Faro, Portugal, 21–22 November 2019; pp. 17–24.
20. Kostov, G.; Wolfartsberger, J. Designing a Framework for Collaborative Mixed Reality Training. *Procedia Comput. Sci.* **2022**, *200*, 896–903. [CrossRef]
21. Brookes, J.; Warburton, M.; Alghadier, M.; Mon-Williams, M.; Mushtaq, F. Studying Human Behavior with Virtual Reality: The Unity Experiment Framework. *Behav. Res. Methods* **2020**, *52*, 455–463. [CrossRef] [PubMed]
22. Bebko, A.O.; Troje, N.F. bmlTUX: Design and Control of Experiments in Virtual Reality and Beyond. *i-Perception* **2020**, *11*. [CrossRef] [PubMed]
23. Maloney, D.; Freeman, G.; Robb, A. A Virtual Space for All: Exploring Children's Experience in Social Virtual Reality. In Proceedings of the 7th Annual Symposium on Computer-Human Interaction in Play (CHI PLAY), Virtual Event, 2–4 November 2020; pp. 472–483.
24. Picard, S.; Botev, J.; Niknam, S. A Dynamic and Scriptable Environment and Framework for Stimulus-Based Cognitive Research in Virtual Reality. In Proceedings of the 2023 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), Sydney, Australia, 16–20 October 2023; pp. 387–392.